

Angular SPA

Case Study



INHALT

- Übersicht + Recap
- Case Studies
 - Component Communication / App Bootstrap
 - Error Handling
 - Filter – Pipe
 - Router / Forms Validation
 - Translate / i18n
- Testing
 - Unitests
 - E2E Tests



Die 2BIT GmbH vereinfacht die technologische Nutzung der Enduser in der Sprache des Kunden auf allen Geräten.

Recap

- Was wisst ihr noch von Angular und TypeScript?



TypeScript



webpack



Angular Advanced

Specific Case Studies



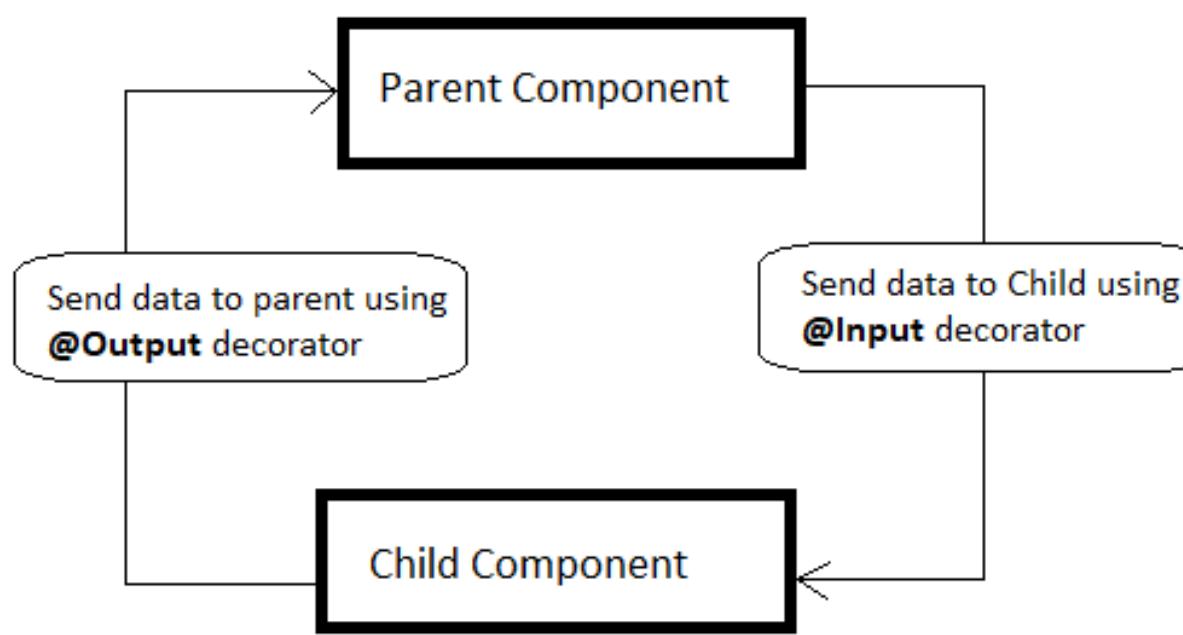
Todos and Todos ...

- Für unsere Übungen bauen wir jeweils immer wieder dieselbe Todo Applikation (kennt ihr ja bereits aus dem Tutorial ;)). Es steht jedoch in jeder Übung wieder ein anderes Detail im Vordergrund auf welches wir uns fokussieren.
- Ihr könnt die Musterlösungen zu allen Übungen die wir zusammen machen hier finden
 - Cloned alle Repos und bringt sie lokal zum Laufen

<https://github.com/rdnscr/2school-angular-solution>

<https://github.com/rdnscr/2school-angular-components>

<https://github.com/rdnscr/2school-angular-advanced>



Component Communication

Component Communication

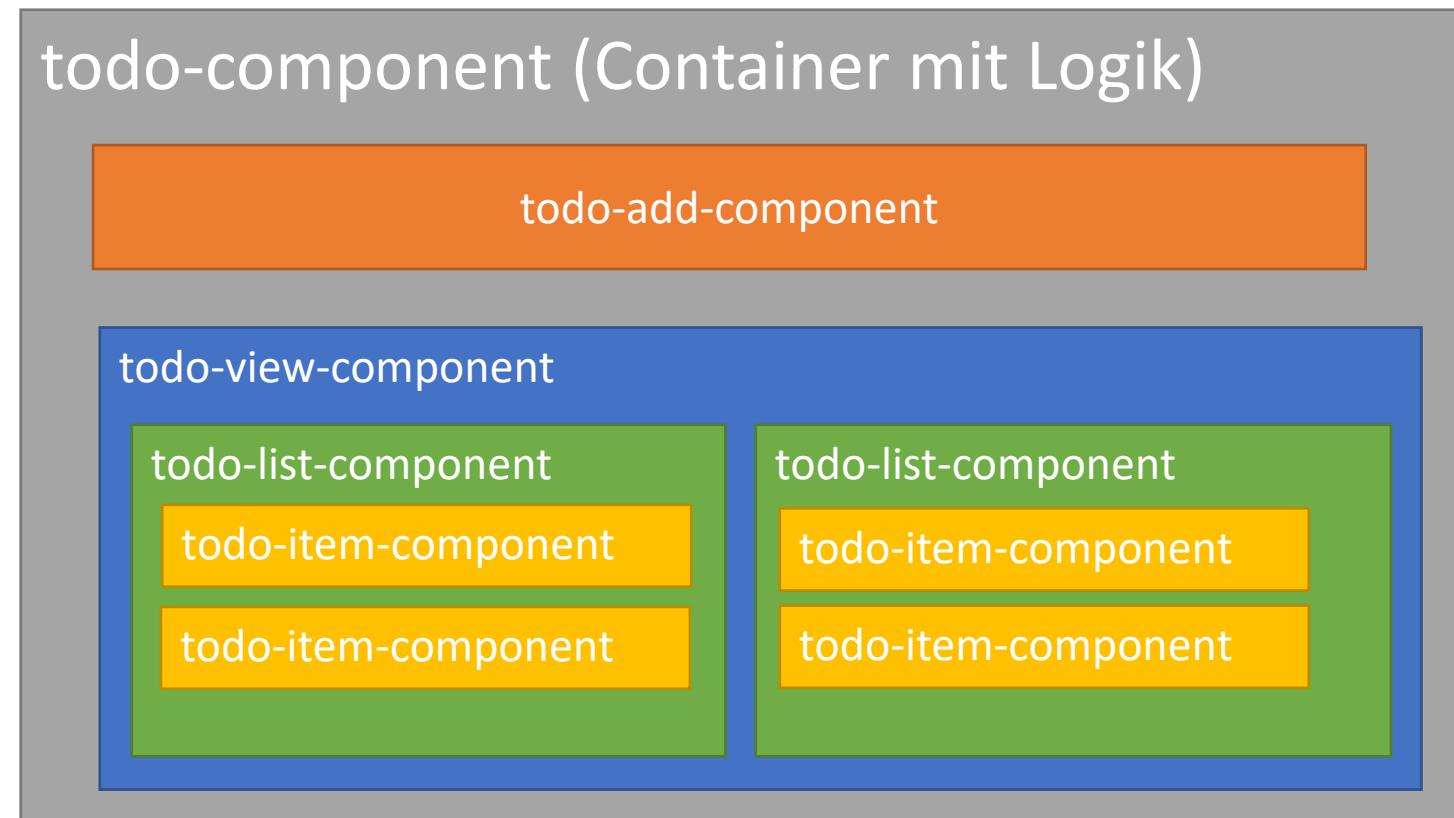
- In Angular versuchen wir in Komponenten zu denken und unsere Seiten entsprechend so zu organisieren.
- Gibt es hier diesbezüglich Optimierungspotential in der aktuellen Ausgangslage?
 - <https://github.com/rdnscr/2school-angular-components>



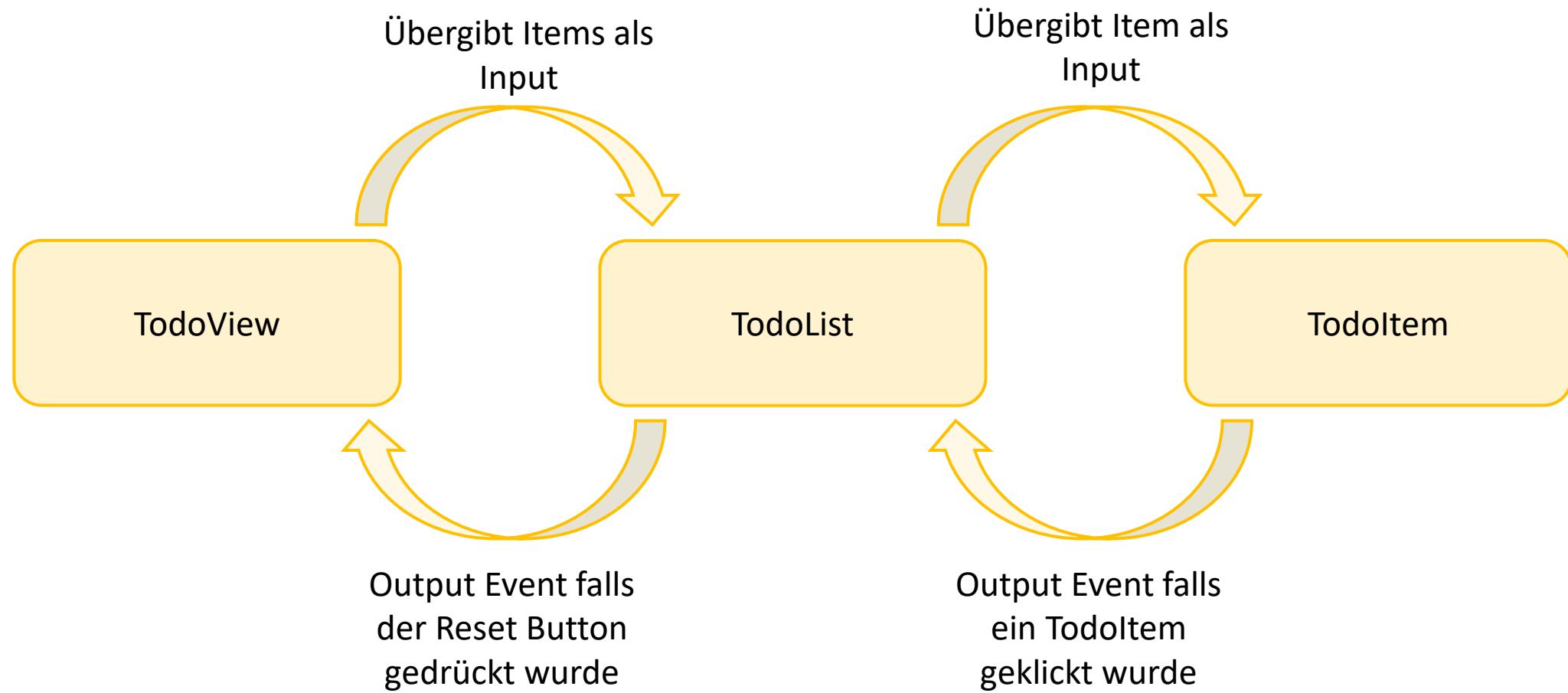
Component Communication

- Durch das saubere erstellen von Komponenten erreichen wir einen hohen Code Reuse auch auf HTML / CSS Level
- Best Practices die wir befolgen um Code zu schreiben, können dadurch auch einfach auf den View Layer (HTML) angewandt werden
- Wir erreichen schlussendlich durch das Erstellen von kleinen unabhängigen Komponenten sauberen und wartbaren Code

Struktur



Component Communication



Component Communication

- Erstellt die entsprechenden Komponenten und führt ein Refactoring durch um die besprochenen Komponenten zu realisieren
 - Die Komponenten sollen miteinander kommunizieren über Inputs und Outputs

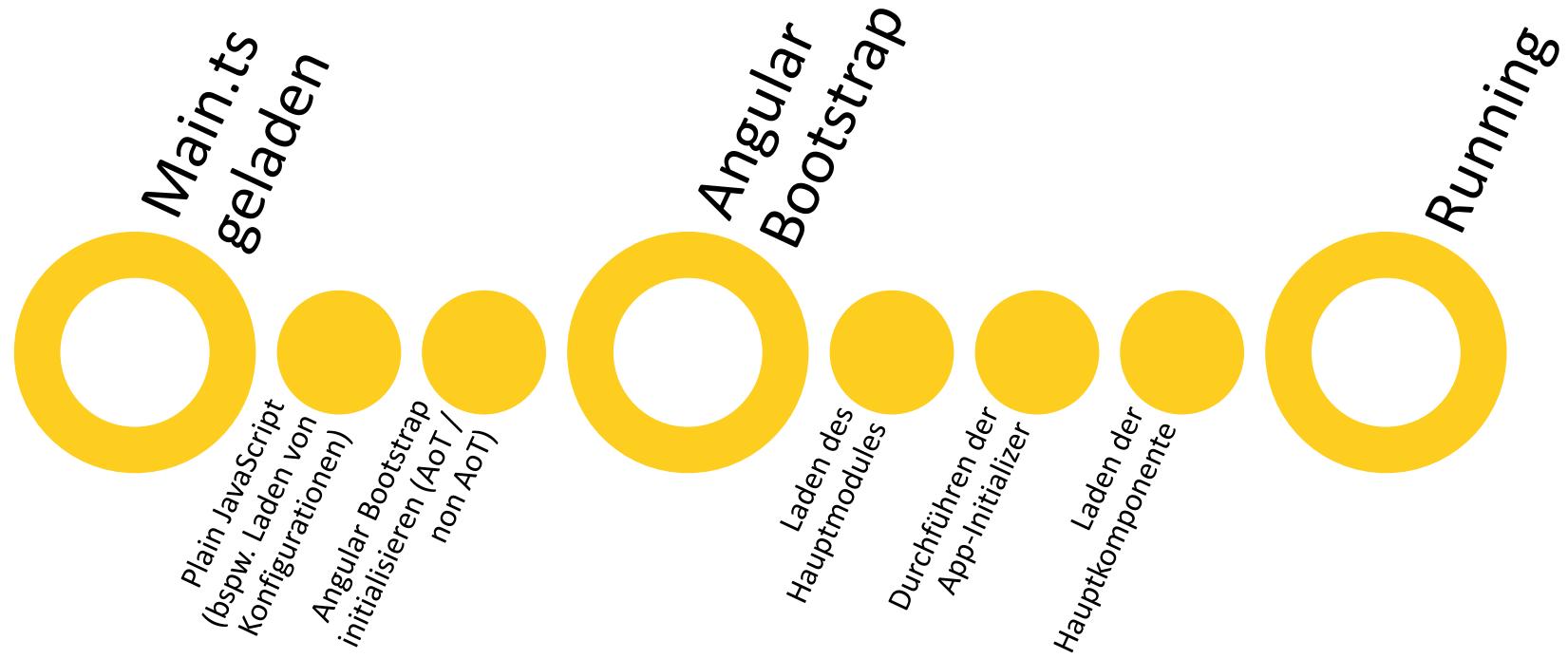
<https://github.com/rdnscr/2school-angular-components>



Boostrapping

```
providers: [
    ConfigServiceService,
    {
        provide: APP_INITIALIZER,
        useFactory: loadConfigurations
        deps: [ConfigServiceService],
        multi: true
    }
],
```

Application Bootstrap



- <https://angular.io/guide/bootstrapping>
- https://angular.io/api/core/APP_INITIALIZER

Application Bootstrap

- Beim Bootstrap muss sich also die Frage gestellt werden: Was möchte ich machen?
 - Initial Daten laden
 - Konfigurationen vom Server laden (könnte auch Styling betreffen)
 - Sonstige Initialisierung
- Der Bootstrap ist nicht zu verwechseln mit Security Konzepten!
 - Security wird über die Guard Funktionalität des Routers gemacht und hat nichts mit Bootstrapping zu tun

Error



An error occurred.



```
[WDS] Live Reloading enabled. client:52
✖ ▶ ERROR Error: Unexpected Error: core.js:4081
    at TodoAddComponent.onAdd (todo-add.component.ts:1
  9)
    at
TodoAddComponent_Template_button_click_8_listener (todo
-add.component.html:9)
    at executeListenerWithErrorHandling (core.js:14296)
    at <anonymous> [monolithicAndDefault / core]
```



Error Handling

Code-Auschnitte

todo.module.ts

```
providers: [TodoService, ErrorLoggerService,
  { provide: ErrorHandler, useClass: ErrorHandlerService },
  DialogService],
```

```
@Injectable()
export class ErrorHandlerService extends ErrorHandler {

  constructor(private logger: ErrorLoggerService) {
    super();
  }

  public handleError(exception: any) {
    if (exception.message && exception.message.length > 0) {
      this.logger.log(exception.message);
    } else {
      this.logger.log('oops something went wrong!');
    }
  }

  super.handleError(exception);
}
```

```
[WDS] Live Reloading enabled. client:52
✖ ▶ ERROR Error: Unexpected Error: core.js:4081
  at TodoAddComponent.onAdd (todo-add.component.ts:1)
  2)
  at
  TodoAddComponent_Template_button_click_8_listener (todo
  -add.component.html:9)
  at executeListenerWithErrorHandling (core.js:14296)
  + import {executeListenerWithErrorHandling} from '@angular/core'
```

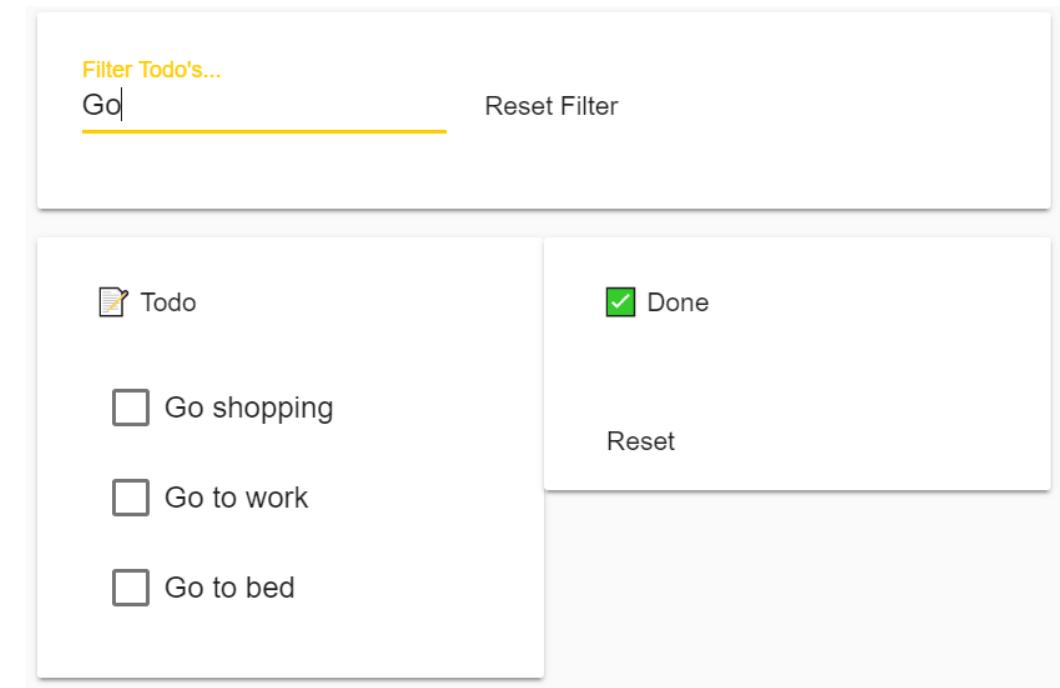


Pipes / Filter



Filter / Pipe - Todo-Filter

- Filtert die sichtbaren TodoItems
- Filter Komponente erstellen inklusive Debounce mit rxjs
- Erstellen einer eigenen Pipe
 - TodoFilterPipe
 - transform – Methode implementieren
 - Todo.Description filtern mit .indexOf()
- Pipe beim Modul registrieren
- Filter Feld hinzufügen
 - Reset Filter Knopf
- Filter Pipe auf der Liste anwenden



Reactive Filtering

- debounceTime(500)
 - 500ms Verzögerung der Eingabe

```
export class TodoFilterComponent {  
  public filter: string;  
  
  @Output()  
  public filterChanged = new EventEmitter<string>();  
  
  private keyPress$ = new Subject<string>();  
  
  constructor() {  
    this.keyPress$.pipe(debounceTime(500)).subscribe((value) => {  
      this.filter = value;  
      this.filterChanged.emit(value);  
    });  
  }  
  
  public valueChanged(value: string) {  
    this.keyPress$.next(value);  
  }  
  
  public resetFilter() {  
    this.valueChanged('');  
  }  
}
```



Routing /
Guards

Router

• Async Routes

- Routes werden ebenfalls über Modul Import im Router registriert
- Man beachte, dass es sich hier um Child routes handelt und daher der Name der Route im app.routes bestimmt wird

```
export const appRoutes: Routes = [
  { path: '', redirectTo: 'todo', pathMatch: 'full' },
  {
    path: 'todo',
    loadChildren: () => import('./todo/todo.module').then(m => m.TodoMvcPlusServiceModule),
    data: { title: 'TODO PWA' }
  }
];
```

- Async Routes

- Im app.routes wird dann das Modul asynchron geladen sobald auf die Route navigiert wird

```
import { Routes } from '@angular/router';
```

```
export const appRoutes: Routes = [
  { path: '', redirectTo: 'todo', pathMatch: 'full' },
  {
    path: 'todo',
    loadChildren: () => import('./todo/todo.module').then(m => m.TodoAdvancedFormsTemplateModule),
    data: { title: 'Todo Template Forms' }
  }
];
```

Pfad zum Module File

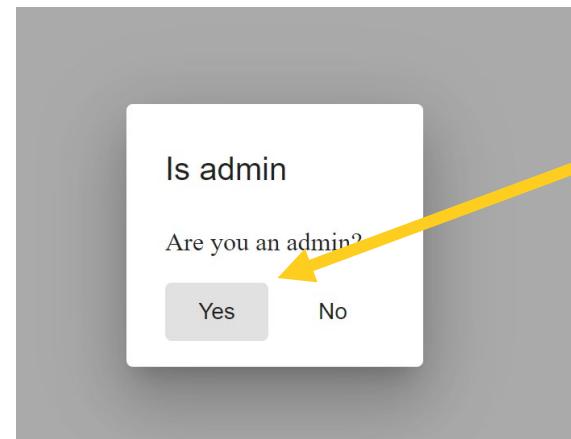
Name des Moduls

```
@NgModule({
  ...
})
export class TodoAdvancedFormsTemplateModule { }
```

Router – canActivate

todo.routes.ts

```
{
  path: '',
  children: [
    {
      path: '',
      component: TodoComponent,
      canActivate: [CanActivateTodoService],
      canDeactivate: [CanDeactivateTodoService],
      resolve: { todos: TodosResolve }
    },
  ],
},
```

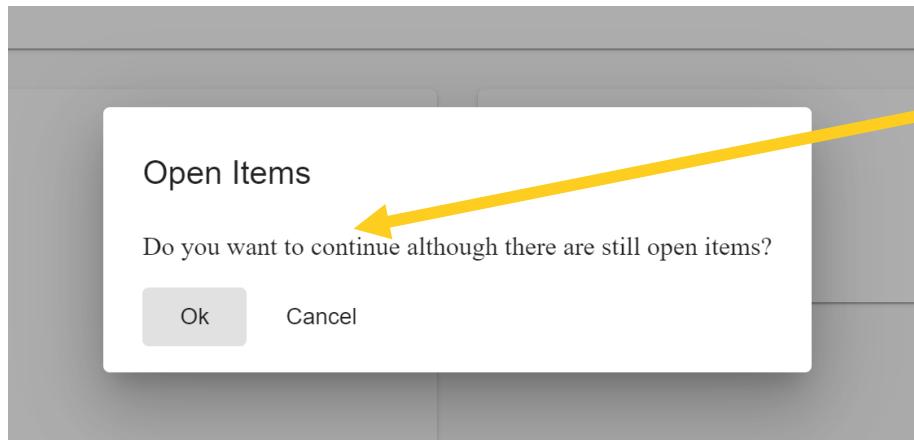


```
@Injectable()
export class CanActivateTodoService implements CanActivate {
  constructor(private snackbar: MatSnackBar, private dialog: MatDialog) {}

  public canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> | boolean {
    const result$ = new Subject<boolean>();
    const dialogRef = this.dialog.open(IsAdminDialogComponent);
    dialogRef.afterClosed().subscribe((result) => {
      if (result === 'yes') {
        result$.next(true);
      } else {
        result$.next(false);
      }
    });
    result$.complete();
  }
}
```

Router – canActivate

```
{
  path: '',
  children: [
    {
      path: '',
      component: TodoComponent,
      canActivate: [CanActivateTodoService],
      canDeactivate: [CanDeactivateTodoService],
      resolve: { todos: TodosResolve }
    },
  ],
},
```



```
@Injectable()
export class CanDeactivateTodoService implements CanDeactivate<TodoComponent>
  constructor(private dialog: MatDialog) { }

}

public canDeactivate(
  component: TodoComponent,
  currentRoute: ActivatedRouteSnapshot,
  currentState: RouterStateSnapshot,
  nextState?: RouterStateSnapshot
): Observable<boolean> {
  const result$ = new Subject<boolean>();

  if (component.todos.filter(item => !item.checked).length > 0) {
    const dialogRef = this.dialog.open(OpenItemsDialogComponent);
    dialogRef.afterClosed().subscribe(result => {
      if (result === 'ok') {
        result$.next(true);
      } else {
        result$.next(false);
      }
    });
  }

  return result$;
}
```

Router - resolve

```

{
  path: '',
  children: [
    {
      path: '',
      component: TodoComponent,
      canActivate: [CanActivateTodoService],
      canDeactivate: [CanDeactivateTodoService],
      resolve: { todos: TodosResolve }
    },
  ],
  @Injectable()
  export class TodosResolve implements Resolve<TodoItem[]> {

    constructor(private todoService: TodoService) { }

    public resolve(route: ActivatedRouteSnapshot): Observable<TodoItem[]> {
      return this.todoService.load();
    }
  }
}

```

todo.component.ts

```

public ngOnInit() {
  // this.todoService.load();
}

```

Security – JWT Token - DEMO

Username: test
Password: test

Angular JWT Login Example

Username

Password

Login

- Basierend auf:
 - <https://jasonwatmore.com/post/2019/06/22/angular-8-jwt-authentication-example-tutorial#auth-guard-ts>
 - <https://github.com/cornflourblue/angular-8-jwt-authentication-example>

HTTP Interceptor

```
@Injectable({ providedIn: 'root' })
export class UserService {
  constructor(private http: HttpClient) { }

  getAll() {
    return this.http.get<User[]>(`${environment.apiUrl}/users`);
  }
}
```

```
@Injectable()
export class JwtInterceptor implements HttpInterceptor {
  constructor(private authenticationService: AuthenticationService) { }

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    // add authorization header with jwt token if available
    let currentUser = this.authenticationService.currentUserValue;
    if (currentUser && currentUser.token) {
      request = request.clone({
        setHeaders: {
          Authorization: `Bearer ${currentUser.token}`
        }
      });
    }
    return next.handle(request);
  }
}
```

Registration Interceptor im Modul

```
],
providers: [
  { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
  { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },
  // provider used to create fake backend
  fakeBackendProvider
], jwatmore, a year ago • initial commit
```

```
@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
  constructor(private authenticationService: AuthenticationService) { }

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return next.handle(request).pipe(catchError(err => {
      if (err.status === 401) {
        // auto logout if 401 response returned from api
        this.authenticationService.logout();
        location.reload(true);
      }
      const error = err.error.message || err.statusText;
      return throwError(error);
    }));
  }
}
```

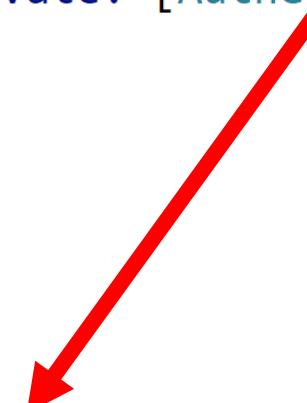
CanActivate – im Beispiel

```
const routes: Routes = [
  { path: '', component: HomeComponent, canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },

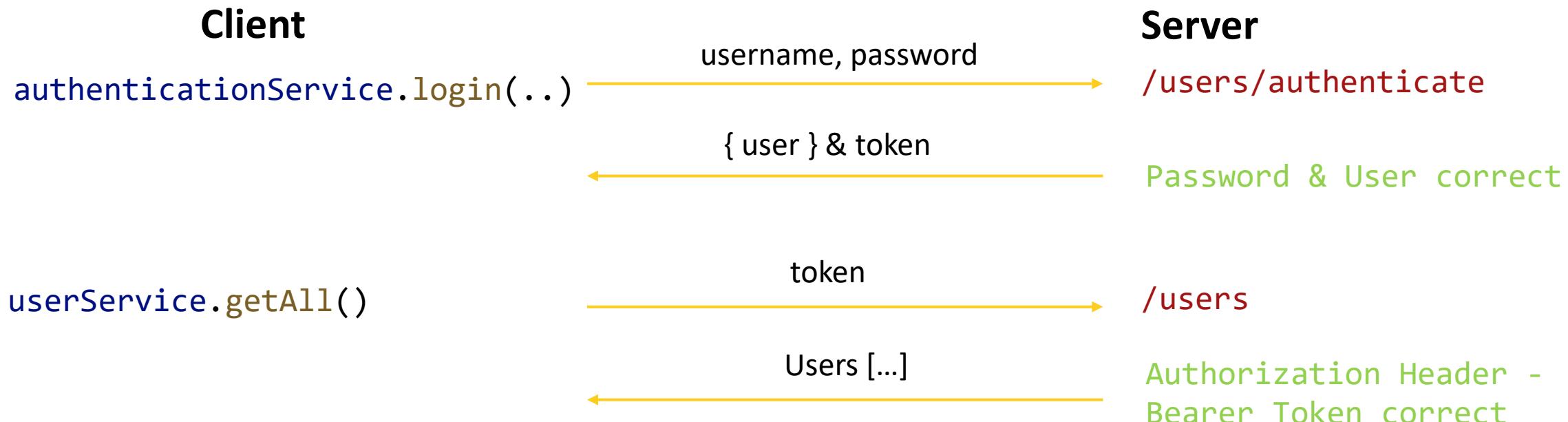
  @Injectable({ providedIn: 'root' })
  export class AuthGuard implements CanActivate {
    constructor(
      private router: Router,
      private authenticationService: AuthenticationService
    ) { }

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
      const currentUser = this.authenticationService.currentUserValue;
      if (currentUser) {
        // logged in so return true
        return true;
      }

      // not logged in so redirect to login page with the return url
      this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });
      return false;
    }
}
```



Authentisierung & Token - DEMO



Forms & Validation

Forms & Validation

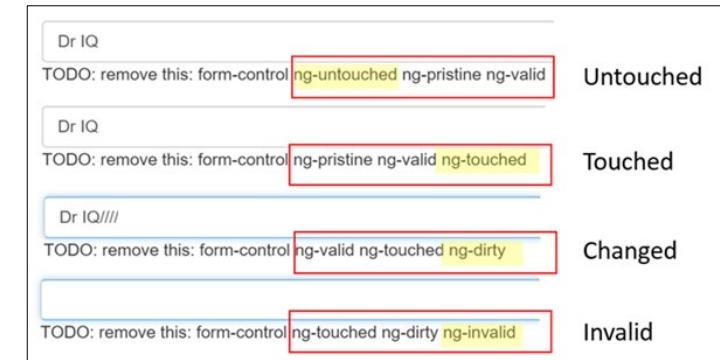
- Es gibt zwei verschiedene Arten wie in Angular Forms erstellt werden können
 - Template Driven
 - Reactive Forms
- Bei beiden Varianten werden einige Angular Direktiven und Services benötigt

Forms & Validation – Template Driven

- Die gesamten Validationen werden im HTML ausprogrammiert
- Hierzu werden folgende Direktiven benötigt
 - ngForm, ngModel
- Template Driven bedeutet
 - Ähnlich wie im Angular 1
 - ngModel für Two Way Binding
 - ngModel einer Variable zugewiesen um Validierungen zu überprüfen
 - Validation im HTML
 - Meldungen im HTML
- Teil von *FormsModule* in '@angular/forms'

Forms & Validation – Template Driven

- Für die Validation innerhalb des HTML stehen folgende Informationen für die Verschiedenen ngControls zur Verfügung
 - dirty <-> pristine (verändert / unverändert)
 - touched <-> untouched (Control wurde bereits fokussiert / nicht fokussiert)
 - valid <-> invalid (Control gültig oder ungültig)
- CSS Klasse je nach Status



<https://angular.io/guide/forms>

Forms & Validation – Template Driven

```
<form (ngSubmit)="onSubmit()" #heroForm="ngForm">
  <div class="form-group">
    <label for="name">Name</label>
    <input type="text" class="form-control" id="name"
           required
           [(ngModel)]="model.name" name="name"
           #name="ngModel">
    <div [hidden]="name.valid || name.pristine"
         class="alert alert-danger">
      Name is required
    </div>
  </div>
```

Forms & Validation – Template Driven

- Die Validierungsmeldungen werden im Code der Komponente generiert
 - Validierungsmeldungen im Controller durch Subscription auf valueChange
 - Es wird dadurch nicht so viel unnötiges HTML geschrieben um jede Fehlermöglichkeit abzufangen
 - Die Möglichkeiten um dynamische Fehlermeldungen auszugeben sind im JavaScript / TypeScript grösser

Forms & Validation – Template Driven

```
<h1>Template-Driven Form</h1>
<form #heroForm="ngForm" appIdentityRevealed>
  <div [hidden]="heroForm.submitted">
    <div class="cross-validation" [class.cross-validation-error]="heroForm.errors | async">
      <div class="form-group">
        <label for="name">Name</label>
        <input id="name" name="name" class="form-control"
          required minlength="4" appForbiddenName="bob"
          [(ngModel)]="hero.name" #name="ngModel" >

        <div *ngIf="name.invalid && (name.dirty || name.touched)"
          class="alert alert-danger">
          <div *ngIf="name.errors.required">
            Name is required.
          </div>
        </div>
      </div>
    </div>
  </div>
</form>

heroForm: NgForm;
@ViewChild('heroForm') currentForm: NgForm;

ngAfterViewChecked() {
  this.formchanged();
}

formchanged() {
  if (this.currentForm === this.heroForm) { return; }
  this.heroForm = this.currentForm;
  if (this.heroForm) {
    this.heroForm.valueChanges
      .subscribe(data => this.onValueChanged(data));
  }
}
```

Forms & Validation – Template Driven

```
onvalueChanged(data?: any) {
  if (!this.heroForm) { return; }
  const form = this.heroForm.form;

  for (const field in this.formErrors) {
    // clear previous error message (if any)
    this.formErrors[field] = '';
    const control = form.get(field);

    if (control && control.dirty && !control.valid) {
      const messages = this.validationMessages[field];
      for (const key in control.errors) {
        this.formErrors[field] += messages[key] + ' ';
      }
    }
  }
}
```

Forms & Validation – Reactive Forms

- Es wird alles innerhalb des Controllers definiert
 - Erstellen der Form im Code (*new FormGroup({«Forminhalt»})*)
 - Kein ngModel Binding mehr
 - Klarer Object Tree
 - Validation im Controller
 - Control gebunden auf den Control Namen
 - Synchrone Ablauf
 - Validationen können in UnitTests verifiziert werden
- Teil von ReactiveFormsModule in '@angular/forms'

Beschreibungen der wichtigsten Klassen

- `AbstractControl`: Abstrakte Basisklasse für `FormControl`, `FormGroup` und `FormArray`
- `FormControl`: Trackt den Wert und Gültigkeit eines einzelnen Form Control (Input, Select usw.)
- `FormGroup`: Trackt Wert und Gültigkeit einer Gruppe von `AbstractControls` und deren Kinder.
- `FormArray`: Trackt Wert und Gültigkeit eines indexierten Arrays von `AbstractControls`

Forms & Validation – Reactive Forms

```

    heroForm: FormGroup;

    ngOnInit(): void {
      this.heroForm = new FormGroup({
        'name': new FormControl(this.hero.name, [
          Validators.required,
          Validators.minLength(4),
          forbiddenNameValidator(/bob/i)
        ]),
        'alterEgo': new FormControl(this.hero.alterEgo,
          asyncValidators: [this.alterEgoValidator.validator],
          updateOn: 'blur'
        ),
        'power': new FormControl(this.hero.power, Validators.required),
        ... { validators: identityRevealedValidator })
    }
  }

```

```

<form [formGroup]="heroForm" #formDir="ngForm">
  <div [hidden]="formDir.submitted">
    <div class="cross-validation" [class.cross-validation-error]="hero.name.errors.length > 0">
      <div class="form-group">
        <label for="name">Name</label>
        <input id="name" class="form-control" formControlName="name" required>
        <div *ngIf="name.invalid && (name.dirty || name.touched)" class="alert alert-danger">
          Name is required.
        </div>
      </div>
    </div>
  </div>
</form>

```

- <https://angular.io/generated/live-examples/form-validation/stackblitz.html>

Custom Validators

Template Driven

```
<input id="name" name="name" class="form-control"
       required minlength="4" forbiddenName="bob"
       [(ngModel)]="hero.name" #name="ngModel" >
```

shared/forbidden-name.directive.ts (directive)

```
@Directive({
  selector: '[appForbiddenName]',
  providers: [{provide: NG_VALIDATORS, useExisting: ForbiddenValidatorDirective, multi: true}]
})
export class ForbiddenValidatorDirective implements Validator {
  @Input() forbiddenName: string;

  validate(control: AbstractControl): {[key: string]: any} {
    return this.forbiddenName ? forbiddenNameValidator(new RegExp(this.forbiddenName, 'i'))(control)
      : null;
  }
}
```

reactive/hero-form-reactive.component.ts (validator functions)

```
this.heroForm = new FormGroup({
  'name': new FormControl(this.hero.name, [
    Validators.required,
    Validators.minLength(4),
    forbiddenNameValidator(/bob/i) // <- Here's how you pass in the cust
  ]),
  'alterEgo': new FormControl(this.hero.alterEgo),
  'power': new FormControl(this.hero.power, Validators.required)
});
```

shared/forbidden-name.directive.ts (forbiddenNameValidator)

```
/** A hero's name can't match the given regular expression */
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {
  return (control: AbstractControl): {[key: string]: any} => {
    const forbidden = nameRe.test(control.value);
    return forbidden ? {'forbiddenName': {value: control.value}} : null;
  };
}
```

Reactive

- Was ist der Unterschied zwischen Model-Driven und Template Driven?
- Was ist der Vorteil von Model Driven gegenüber Template Driven?
- Was ist der Unterschied zwischen dem formControl und dem formControlName?



Forms & Validation

- Schaut euch das folgende Beispiel an
 - <https://angular.io/guide/form-validation>
- Öffnet das Live Beispiel
 - <https://angular.io/generated/live-examples/form-validation/stackblitz>
- Reflektiert die Unterschiede zwischen Template Driven und Reactive Forms
 - Versucht zudem ein neues Feld in den jeweiligen Versionen einzubauen, welches entsprechend validiert wird



Forms & Validation

- Schritte zur Lösungserarbeitung
 - Form Tag hinzufügen
 - Input mit entsprechenden HTML Validationen ausstatten
 - NgModel Binding erstellen und dieses auf Gültigkeit überprüfen um Fehler anzuzeigen

Lösung Template-Driven

```
<mat-form-field>
  <input id="description" name="description"
    required minlength="3"
    matInput placeholder="Todo Beschreibung"
    [(ngModel)]="descriptionText"
    #descriptionModel="ngModel">
  <mat-error *ngIf="descriptionModel.invalid &&
    (descriptionModel.dirty || descriptionModel.touched)"
    class="alert alert-danger">
    <div *ngIf="descriptionModel.errors.required">
      | | Description is required.
    </div>
    <div *ngIf="descriptionModel.errors.minlength">
      | | Description must be at least 3 characters long.
    </div>
  </mat-error>
</mat-form-field>
```

- Reactive Forms
 - Wir wollen beim Einfügen für ein neues Item eine Reactive Form bauen, die auch gleich auf Required und mindestens 3 Zeichen validiert



Forms & Validation

- Schritte zur Lösungserarbeitung
 - Form Tag hinzufügen
 - FormGroup mit FormControls erstellen
 - FormControl erstellen mit den entsprechenden Validatoren
 - Input an entsprechendes FormControl binden
 - formControlName Direktive dazu verwenden

I18N

HOLA

GUTEN
TAG

BONJOUR

Hi

Translate (i18n)

- Angular i18n
 - Built in Angular CLI
 - Internationalisierung mit vielen Features

```
ng add @angular/localize
```

```
i18n-x="|<description>@@<id>"
```

```
src/app/app.component.html
```

```
<h1 i18n="An introduction header for this sample">Hello i18n!</h1>
```

```
src/app/app.component.html
```

```
<img [src]="logo" i18n-title title="Angular logo" />
```

Details

- <https://angular.io/guide/i18n>

I18n – Translate Singular/Plural

src/app/app.component.html

```
<span i18n>Updated {minutes, plural, =0 {just now} =1 {one minute ago} other {{minutes}} minutes ago}</span>
```

- =0 (or any other number)
- zero
- one
- two
- few
- many
- other

I18n – Select alternative text

src/app/app.component.html

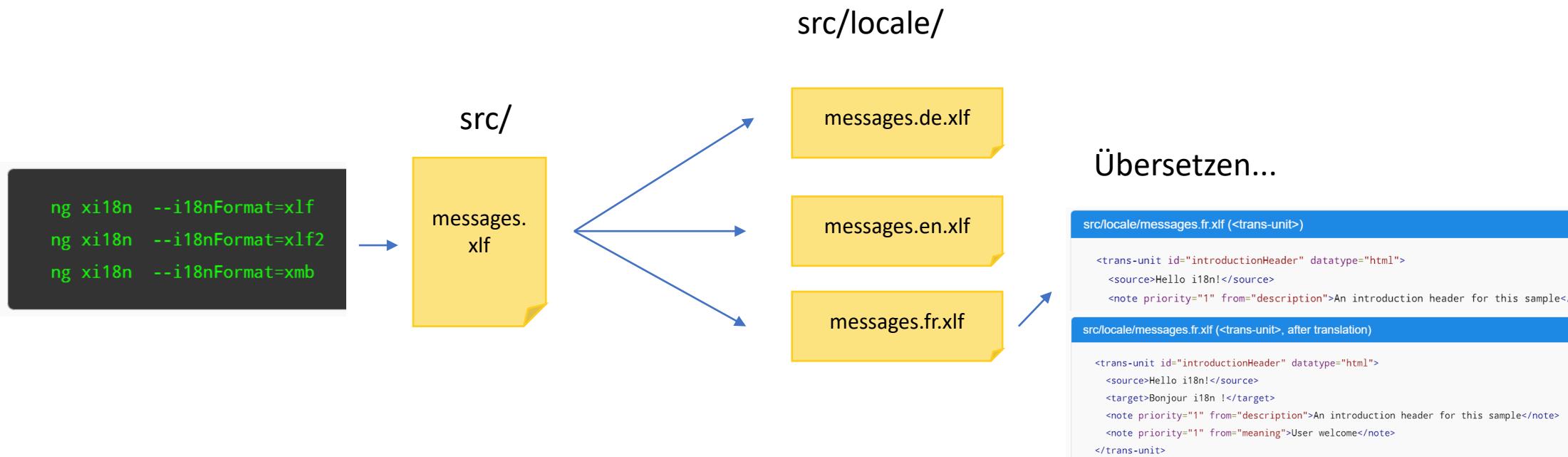
```
<span i18n>The author is {gender, select, m {male} f {female} o {other}}</span>
```

Kombiniert select & plural

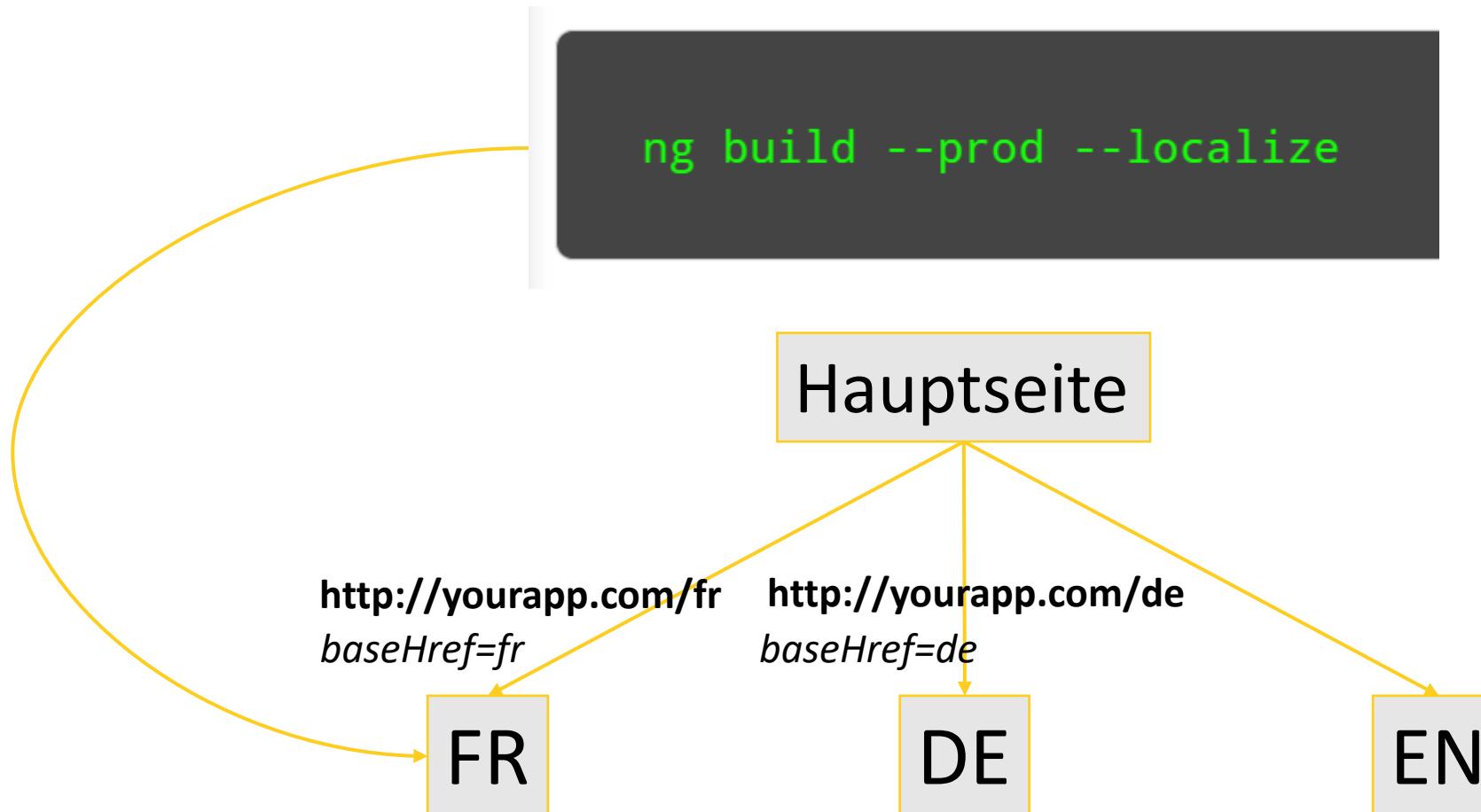
src/app/app.component.html

```
<span i18n>Updated: {minutes, plural,
  =0 {just now}
  =1 {one minute ago}
  other '{{minutes}} minutes ago by {gender, select, m {male} f {female} o {other}}}>
</span>
```

I18n - Prozess



I18n - Deployment



angular.json

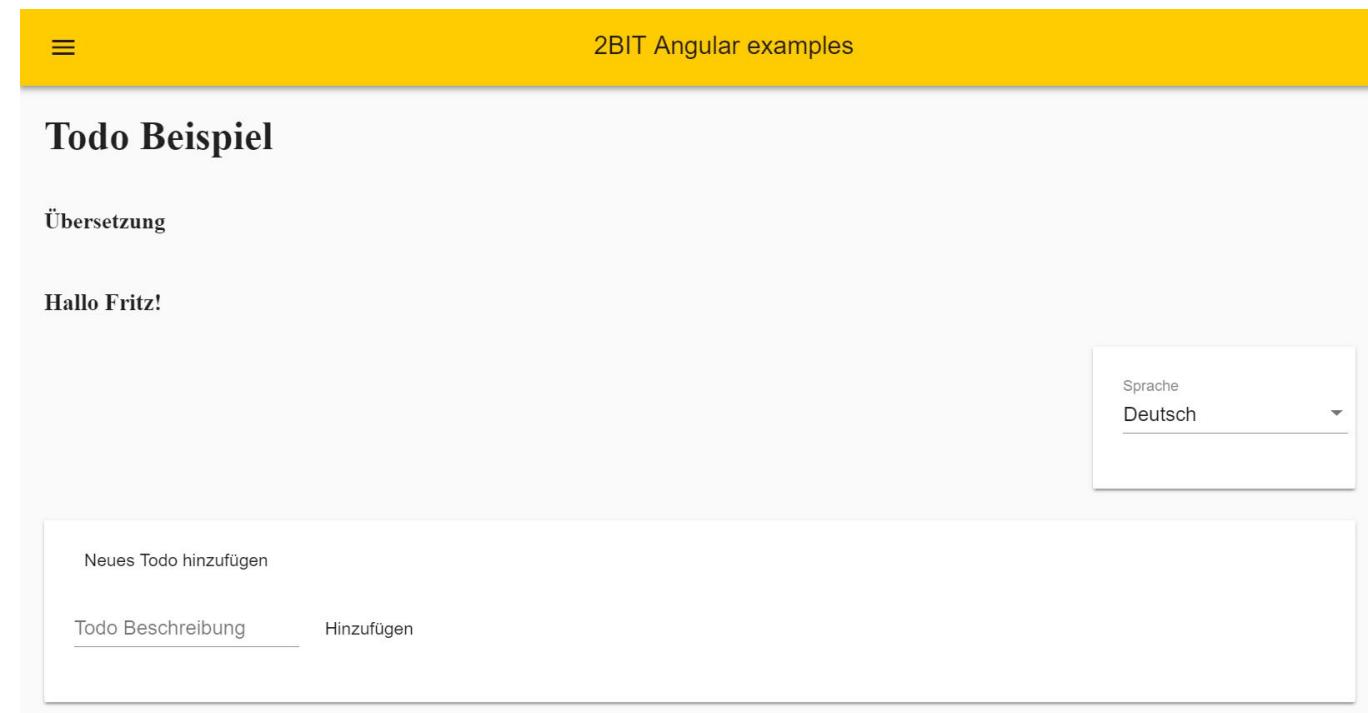
```
"projects": {  
  "angular.io-example": {  
    ...  
    "i18n": {  
      "sourceLocale": "en-US",  
      "locales": {  
        "fr": "src/locale/messages.fr.xlf"  
      } ...  
    },  
    "architect": {  
      ...  
    }  
  }  
}
```

Translate – ngx-translate

- Key / Value – Übersetzung
- Parameterübersetzung möglich
- Missing Translation Handler
- No Plural / No Select
- Easy Service Translation

```
translate.get('HELLO', {value: 'world'}).subscribe((res: string) => {
  console.log(res);
  //=> 'hello world'
});
```

- Plugin: Localize Router
- Plugin: .po usage
- ...



Translate – ngx-translate

- Einfache Übersetzung mit dem Key <-> Value Prinzip

```
todo-list.component.html ×
1  <div class="flex">
2    <md-card>
3      <md-card-header>
4        <md-card-title>{{'TODO' | translate}}</md-card-title>
5      </md-card-header>
6      <md-card-content>
7        <md-list>

<div class="content">
  <h1 translate>TODO EXAMPLE</h1>
  <h2 translate>TRANSLATE</h2>

  <h3>{{ 'HELLO' | translate:param }}</h3>
  ...<lang-switch></lang-switch>
```

```
@NgModule({
  imports: [CommonModule, FormsModule, ReactiveFormsModule, MdCardModule, MdCheckboxModule, todoModule],
  providers: [TranslateModule.forChild({
    loader: {
      provide: TranslateLoader,
      useFactory: HttpLoaderFactory,
      deps: [HttpClient]
    }
  })],
  declarations: [TodoAddComponent, TodoComponent]
})
```

de.json	en.json
1 {	1 {
2 "HELLO":	2 "HELLO": "hello {{value}}!",
3 "TODO_DE	3 "TODO_DESCRIPTION": "Todo Description",
4 "ADD_ITE	4 "ADD_ITEM": "Add a new Item",
5 "Languag	5 "LANGUAGE": "Language",
6 "ADD" :	6 "ADD" : "Add",
7 "TODO_EX	7 "TODO_EXAMPLE": "Todo Example",
8 "TRANSLA	8 "TRANSLATE": "Translate",
9 "TODO":	9 "TODO": "Todo",
10 "DONE":	10 "DONE": "Done",
11 "RESET":	11 "RESET": "Reset"
12 }]	12 }]

- TranslateHttpLoader => /assets/i18n/[lang].json*

Übung Translate - ngx-translate

- Übersetze die Todo-App in Deutsch und Englisch
 - Installiere @ngx-translate an den gegebenen Punkten
 - Registriere das Translate Modul mit den entsprechenden Loader
 - Achtung nicht im Root!
 - Erstelle dafür zwei .json – Files in src\assets\i18n
 - Füge die translate Direktiven an den benötigten Punkten ein
- Drop Down für Language switch -> über TranslateService
- Wie würde aus einem Service eine Übersetzung aufgerufen?
- Source Code:
 - <https://github.com/ngx-translate/core#usage>

Fragen zu ngx-translate

- Wie wird die initiale Sprache festgelegt?
 - Wie müsste ich vorgehen um die Defaultsprache zu ändern?
- Wie wird die Sprache gesetzt über das DropDown?
- Wieso werden die JSON-Files im assets-Ordner abgelegt?
- Wie wird der Parameter für die Übersetzung von 'HELLO' zusammengestellt?

Testing

Angular Unit Testing



Unitests

- Ein Unitest besteht immer aus drei verschiedenen Schritten (AAA)
 - Arrange (Aufsetzen des Tests)
 - Act (Ausführen des Tests)
 - Assert (Verifikation des Testergebnisses)
- Es gibt wie zwei grundsätzliche Ansätze wie wir Angular Klassen testen können
 - Klassisch mit Jasmine (schreiben alles selbst)
 - Benützung der Angular Test Suite um Komponenten / Services innerhalb des Angular Ökosystems zu testen

Unitests - Angular

- Aufsetzen des « TestBed » um das Dependency Injection System entsprechend für den Test vorzubereiten

```
beforeEach(async(() => {
  TestBed.configureTestingModule({
    imports: [HttpModule],
    providers: [
      TodoService,
      { provide: XHRBackend, useClass: MockBackend }
    ]
  })
  .compileComponents();
}));
```

Unitests - Angular

- Konfiguration des Mocks welches als HTTP Service verwendet wird

```
beforeEach(inject([Http, XHRBackend], (http: Http, be: MockBackend) => {
  backend = be;
  service = new TodoService(http);
  fakeTodos = makeTodoData();
  let options = new ResponseOptions({ status: 200, body: { data: fakeTodos } });
  response = new Response(options);
}));

it('should have expected fake todos (then)', async(inject([], () => {
  backend.connections.subscribe((c: MockConnection) => c.mockRespond(response));
}))
```

Unitests - Angular

- Schreiben eines entsprechenden Tests inklusive der Assertion

```
it('should have expected fake todos (then)', async(inject([], () => {
  backend.connections.subscribe((c: MockConnection) => c.mockRespond(response));

  service.load().toPromise()
    .then((todos: any) => {
      expect(todos.data.length).toBe(fakeTodos.length,
        'should have expected no. of todos');
    });
})));
```

Unitests - Klassisch

- Die Unitests werden klassisch inklusive der verwendeten Mocks selbst geschrieben
 - Dies ist in einer dynamischen Sprache wie JavaScript sehr einfach möglich, da jederzeit on the Fly JavaScript Objekte mit entsprechenden Funktionen erstellt werden können
 - Asynchronität wird innerhalb des Tests entfernt und durch synchrone Funktionen ersetzt

Unitests - Klassisch

- Aufsetzen des Tests und erstellen der entsprechenden Mocks
 - Mocks können am einfachsten mit Spys von Jasmine erstellt werden
 - Mit einem Spy kann nicht nur einfach ein Rückgabewert etc. bestimmt werden, sondern es kann im Test auch verifiziert werden, ob die Funktion effektiv auch aufgerufen wurde

```
// Arrange
const fakeHttp = {get: () => {}} as any;
let observerFake: Observer<any>;
const obs = new Observable((observer) => {
  observerFake = observer;
});
const items = makeTodoData();

spyOn(fakeHttp, 'get').and.returnValue(obs);
const sut = new TodoService(fakeHttp);
```

Unitests - Klassisch

- Sobald der Test entsprechend vorbereitet ist, kann die Funktion welche getestet werden soll aufgerufen werden

```
// Act
let result;
sut.load().subscribe(resultFromService) => {
  result = resultFromService;
});
observerFake.next({json: () => items});
observerFake.complete();
```

Unitests - Klassisch

- Der letzte Schritt ist das Überprüfen des Testresults gegen das erwartete Resultat
 - Mit Spys kann zusätzlich auch noch der Flow innerhalb der Funktion verifiziert werden.
 - Wurden beispielsweise alle Funktionen eines Services der verwendet werden soll aufgerufen?

```
// Assert
expect(fakeHttp.get).toHaveBeenCalled();
expect(result).toBe(items);
```

Unittest

- Schreibt nun den Unittest für den Service welcher verwendet wird um die Todo's zu laden (todo.service.ts)
- Testet diesen Service einmal mit der klassischen Methode und einmal mit dem Angular Test Environment
- Wie dies gemacht werden könnte findet ihr in den Folien oder auf angular.io
 - <https://angular.io/guide/testing>



<https://github.com/rdnscr/2school-angular-advanced>

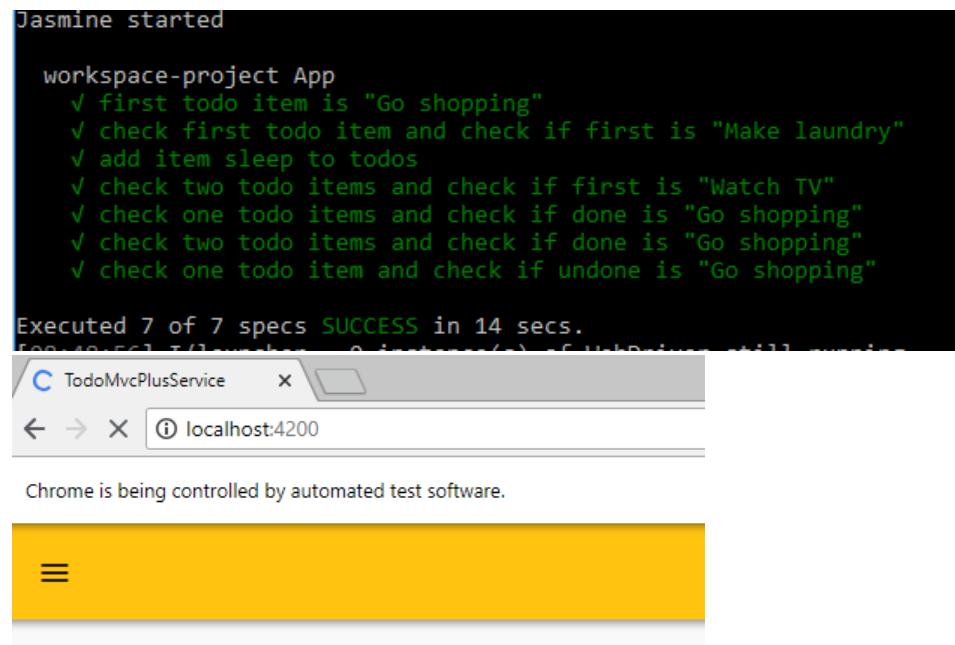
End to End testing

- E2E Testing ermöglicht es die Applikation durch die Simulation eines Benutzers vertikal (alle Layers) zu testen
- E2E Tests sind eher aufwändig zu schreiben
 - Erhöhen aber die Stabilität der Software massiv
 - Sind reproduzierbar und können manuelle Tests reduzieren
 - Sollten mindestens für die wichtigsten Use Cases erstellt werden
 - Sollten erst geschrieben werden, sobald die Applikation bereits einen gewissen Reifegrad besitzt
 - Sollten in den Continous Integration Prozess eingebettet sein

E2E Testing - Protractor

- Practical

- ng e2e todo-testing-e2e
 - Runs Test with Protractor



Page Object - app.po.ts

```
getFirstTodoCheckbox() {
  // tslint:disable-next-line:max-line-length
  return element.all(by.css('mat-checkbox')).first();
}

getFirstDoneTodo() {
  // tslint:disable-next-line:max-line-length
  return element.all(by.xpath('//todo-list[2]//todo-item')).first();
}
```

E2E Spec - app.e2e-spec.ts

```
it('add item sleep to todos', () => {
  page.navigateTo();
  page.getAddTodo().sendKeys('Sleep');
  page.getAddTodoButton().click();

  expect(page.getLastTodo().getText()).toContain('Sleep');
});
```